# Intelligent Systems

## Assignment 5.

## Games and Adversarial Search

## Team members

1: A01332485 Constanza Marini Macouzet CEM

2: A01749803 Alicia Huidobro Espejel CEM

3: A01751168 Diana Laura Aguilar Cervantes CEM

## The heuristic evaluation function

### Description

Our evaluation function is:

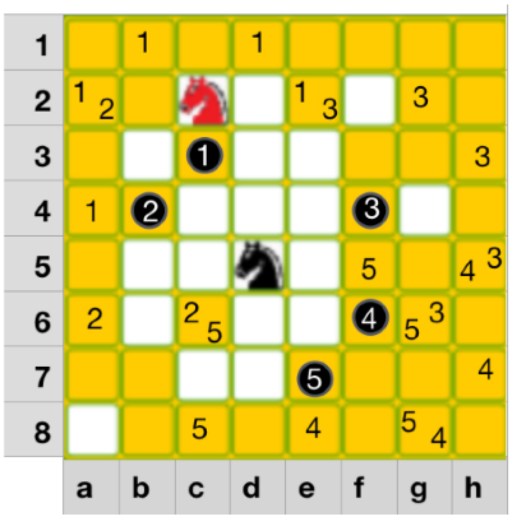
$$f(n) = 3 * next\ move\ blocks\ of\ opponent\ + \dots$$
$$\dots 2 * MAX\ possible\ next\ moves\ +\ \dots$$
$$\dots\ +\ move\ proximity\ to\ opponent$$

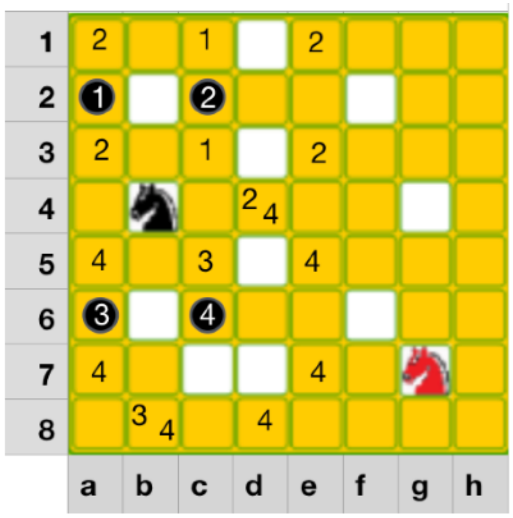
Each feature has a weight value which gives different degree of importance in the evaluation function. These values were computed according to simulations. The evaluation function is as follows:

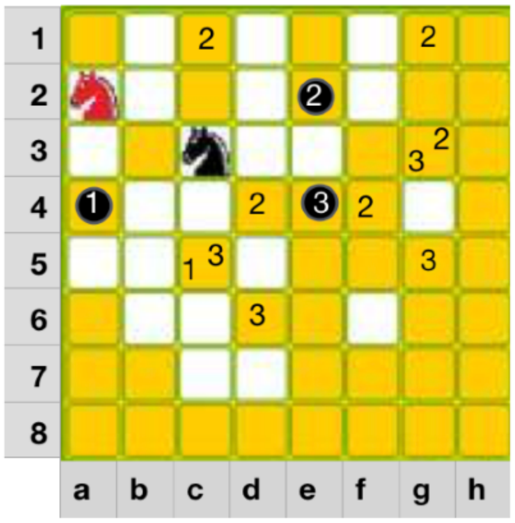
- A possible move of MAX blocks a possible move of MIN: each move that blocks a possible move of MIN weights 3 points.
- The number of possible next-moves that MAX has if it moves to a certain position: each possible move weights 2 points.
- The proximity to the opponent in the move position: it is based on Manhattan distance between the two players and consists of a condition statement: if the opponent is farther than 3 squares, then each extra square will cost 1 point; the same happens if the player is closer than 3 squares. However, it will be compensated with 1 point if it reaches exactly 3 squares away from the opponent.

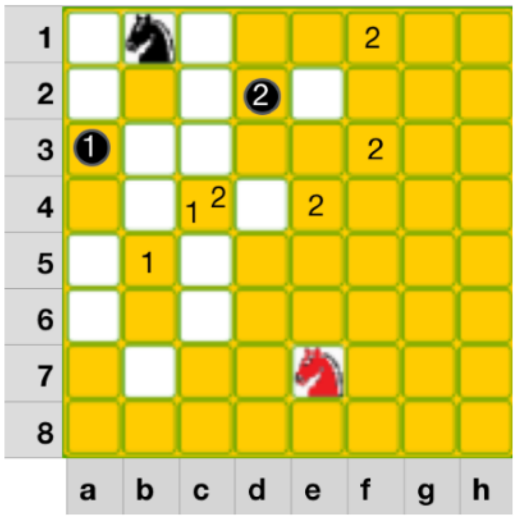
### Evaluation to test boards

Below we show the proposed situations that have to be evaluated (assuming that MAX is the Black Knight). Black circles indicate the possible moves of MAX and each circle indicates the possible next moves with the same number. For example, in SITUATION 1: the ❶ indicates one possible move *(3, c)* and each *1* on the board indicates possible moves from that position *(3, c)*.


Situation 1. MAX moves


Situation 2. MIN moves


Situation 3. MAX moves


Situation 4. MIN moves

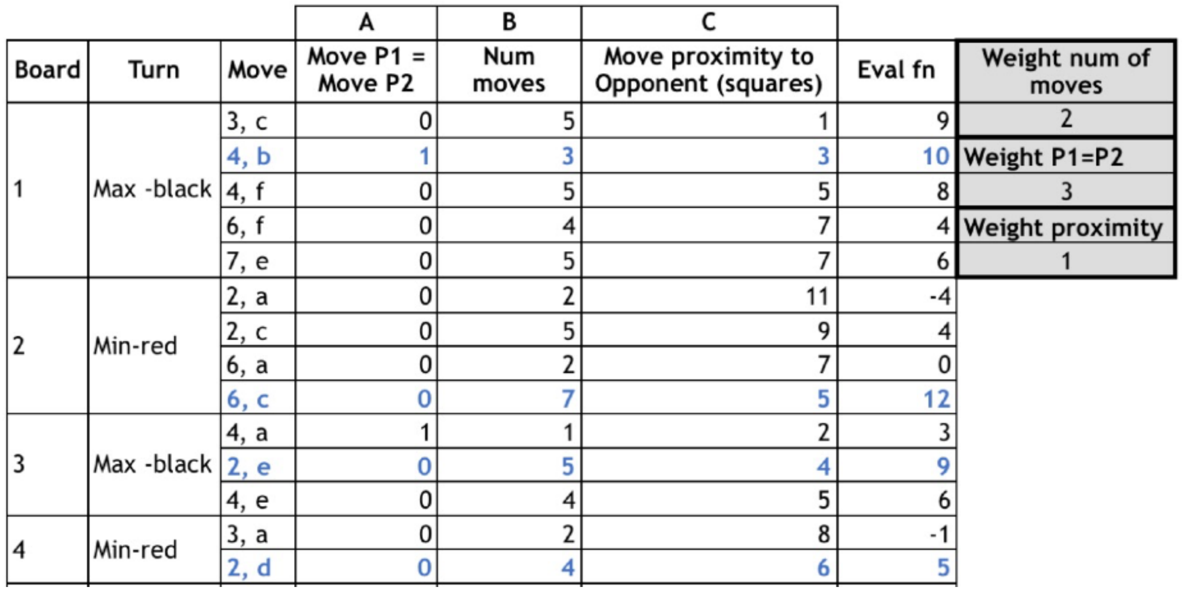
In the next table we show the results of manually applying the evaluation function, considering:

*A = next move blocks move of opponent*

*B = MAX possible next-moves*

*C = move proximity to opponent*

The suggested movement, based on our evaluation function, is indicated in blue.



| | | | A | B | C | | |
|---|---|---|---|---|---|---|---|
| Board | Turn | Move | Move P1 = Move P2 | Num moves | Move proximity to Opponent (squares) | Eval fn | Weight num of moves |
| 1 | Max -black | 3, c | 0 | 5 | 1 | 9 | 2 |
| | | 4, b | 1 | 3 | 3 | 10 | Weight P1=P2 |
| | | 4, f | 0 | 5 | 5 | 8 | 3 |
| | | 6, f | 0 | 4 | 7 | 4 | Weight proximity |
| | | 7, e | 0 | 5 | 7 | 6 | 1 |
| 2 | Min-red | 2, a | 0 | 2 | 11 | -4 | |
| | | 2, c | 0 | 5 | 9 | 4 | |
| | | 6, a | 0 | 2 | 7 | 0 | |
| | | 6, c | 0 | 7 | 5 | 12 | |
| 3 | Max -black | 4, a | 1 | 1 | 2 | 3 | |
| | | 2, e | 0 | 5 | 4 | 9 | |
| | | 4, e | 0 | 4 | 5 | 6 | |
| 4 | Min-red | 3, a | 0 | 2 | 8 | -1 | |
| | | 2, d | 0 | 4 | 6 | 5 | |

## Questions

- **What are the main features of the game that can make a player of any type win or lose?**

We conclude that the main features that could make a player win or lose are:

1. The **number of moves** a player can still make in a specific square (this is one of the main features to be considered). A good player should try to go for squares which have several possible moves; it is important to point out that this feature considers the case when the player is near the edges. As in these squares there are fewer possible moves, squares near the edges should be avoided.
2. At the same time, a good player should aim to **block the opponent's moves** by keeping close to it. This allows the player to reduce the number of moves the opponent has.

- **How easy is it to implement those features in computer systems?**

It would be overwhelming to implement all the possible features that could affect the possibility of winning or losing. Therefore, the initial analysis is crucial in order to select the most relevant features and discard those that can be ignored by implementing others. With a proper problem delimitation and attribute selection, it is still a challenge to convert the rules into Python language but it becomes easier to implement the features.

- **Are they worth considering according to the computational cost of their implementation?**

In the case of Knights Move it is worth to implement the features that typically yield a reasonable solution because it has a reduced number of rules and some rules are included in some way in others. For example: the rule to avoid borders and corners is due to the reduced number of moves in those places, so a rule that considers the number of possible moves implicitly includes it.

- **Is it possible to include strategies within an evaluation function? How?**

Yes, they are included as numerical rules ("rules of thumb"). The weights assigned to the features can also be adjusted in order to induce decisions.

- **In what way is it possible to adjust the weights assigned to the features in the evaluation function?**

First, it was necessary to analyze the behavior of the function while playing, that is, to check if the actions it chose were reasonable. Having done that, we compared the performance of a particular set of parameters by playing against a numb version of the evaluation function itself.

We ran the code several times so as to get some statistics that we used as an indicator of the performance, the more times the evaluation function won, the better. Finally, we found the best two evaluation functions and set them to compete against each other. The winner is the one here presented.